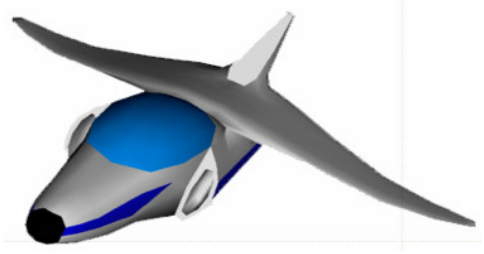


XNA Tutorials

Utah State University
Association for Computing Machinery
XNA Special Interest Group
RB Whitaker
16 October 2007

Index Buffers

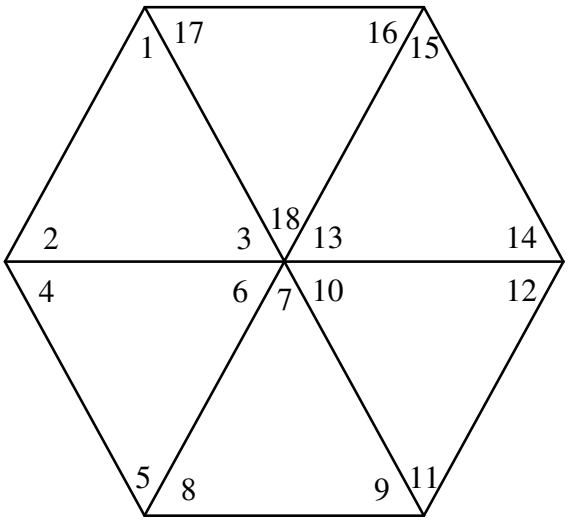
Tutorial 9

Overview

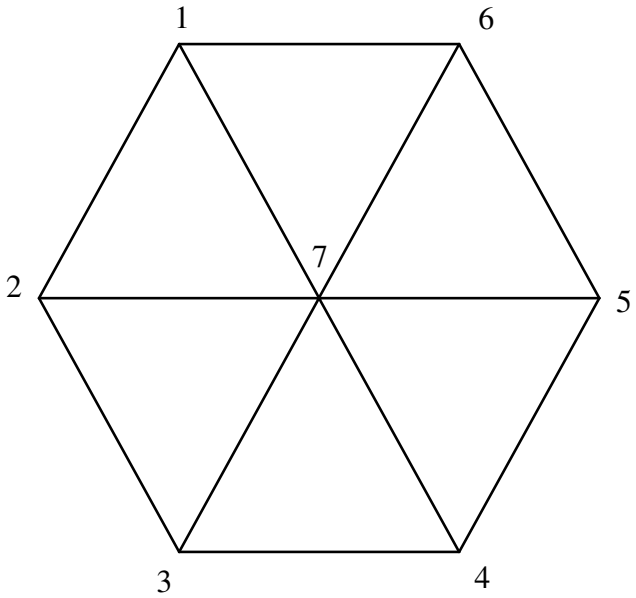
Our next task is to learn how to make our data storage and transfer a little more efficient. This will become extremely important when we start using large models with hundreds, thousands, or even tens of thousands of polygons. You may have noticed in the last tutorial, particularly with the icosahedron, that we seemed to creating a lot of vertices for what we got out of it. That is because we had to define every vertex once for every time it was used. Each of those vertices was used five or six times, so we had a lot of repeats. Index buffers will help us to get around this annoying problem.

Why Do We Want Index Buffers?

What we are about to undertake will take more effort to do, so why do we want to do it? To answer this question, let's first look at the way we have been defining our vertices. Below is a picture of a few triangles. Each vertex of each triangle is numbered in the way that we would have had to number them before.



But if we take a close look, we can see that there are really only seven unique vertices, as illustrated in the picture below.



If we could find a way to only list seven vertices, and then tell each triangle which ones to use, there won't be so many vertices floating around, and we won't have to send so many over to the graphics card either.

This is the idea behind index buffers. With index buffers, we will have two arrays. One will contain the information about each of the vertices, while the other contains the triangle information, specifying with an index, which vertices we are going to use, as shown in the diagram below. The first array, which stores the vertex information, is called the vertex buffer. The second array, which stores indices into the vertex buffer, is called the index buffer.

The Vertex Buffer

The first thing we need to do is set up our vertex buffer. With XNA, there is a built in VertexBuffer class that we can use. First, let's create one of these to use, by adding another class variable to our program. Put the following line just inside the class declaration, where we declare our vertexArray object.

```
private VertexBuffer vertexBuffer;
```

While we're at it, we can remove our declaration for the vertexArray because we won't need that to be a class variable anymore. Remove the line below:

```
private VertexPositionColor[] vertexArray;
```

At this point, we are also done with the SetupVertexArray() method, so you can delete that too. Also delete the call to it in the constructor.

Now, we are going to create a new method for setting up our vertex buffer. So add the method `protected void SetupVertexBuffer()` to your `Asteroids` class. In this method we will need to add the data we want to our new class variable `vertexBuffer`.

We are going to play with the Icosahedron again, since it will be the most interesting. This time, though, we will only have to define each vertex once. You can add the following code to the SetupVertexBuffer method we just added to create this data:

```
// A temporary array, with 12 items in it, because
// the icosahedron has 12 distinct vertices
VertexPositionColor[] vertexArray = new VertexPositionColor[12];

// vertex position information for icosahedron
vertexArray[0].Position = new Vector3(-0.26286500f, 0.0000000f,
            0.42532500f);
vertexArray[1].Position = new Vector3(0.26286500f, 0.0000000f,
            0.42532500f);
vertexArray[2].Position = new Vector3(-0.26286500f, 0.0000000f, -
            0.42532500f);
vertexArray[3].Position = new Vector3(0.26286500f, 0.0000000f, -
            0.42532500f);
vertexArray[4].Position = new Vector3(0.0000000f, 0.42532500f,
            0.26286500f);
vertexArray[5].Position = new Vector3(0.0000000f, 0.42532500f, -
            0.26286500f);
vertexArray[6].Position = new Vector3(0.0000000f, -0.42532500f,
            0.26286500f);
vertexArray[7].Position = new Vector3(0.0000000f, -0.42532500f, -
            0.26286500f);
vertexArray[8].Position = new Vector3(0.42532500f, 0.26286500f,
            0.0000000f);
vertexArray[9].Position = new Vector3(-0.42532500f, 0.26286500f,
            0.0000000f);
vertexArray[10].Position = new Vector3(0.42532500f, -0.26286500f,
            0.0000000f);
vertexArray[11].Position = new Vector3(-0.42532500f, -0.26286500f,
            0.0000000f);

// colors I randomly picked.
// C# has lots defined
vertexArray[0].Color = Color.Red;
vertexArray[1].Color = Color.Orange;
vertexArray[2].Color = Color.Yellow;
vertexArray[3].Color = Color.Green;
vertexArray[4].Color = Color.Blue;
vertexArray[5].Color = Color.Indigo;
vertexArray[6].Color = Color.Purple;
vertexArray[7].Color = Color.White;
vertexArray[8].Color = Color.Cyan;
vertexArray[9].Color = Color.Black;
vertexArray[10].Color = Color.DodgerBlue;
vertexArray[11].Color = Color.Crimson;
```

Now we have to add this data to the vertex buffer we just created. At the bottom of the SetupVertexBuffer that we've been working in, let's add the following two lines of code:

```
vertexBuffer = new VertexBuffer(device, sizeof(float) * 4 *
                   vertexArray.Length, ResourceUsage.WriteOnly,
                   ResourceManagementMode.Automatic);
vertexBuffer.SetData(vertexArray);
```

The first line of code creates a new VertexBuffer object. We need to give it the graphics device we are working with, the amount of space to allocate for it (we have four floats per vertex), followed by the number of vertices in the array, and then a couple of constants to tell XNA how to treat our vertex buffer.

The second line copies the data in vertexArray into the vertexBuffer.

The Index Buffer

Now we need to create our index buffer. Like the vertex buffer, there is a class that we can use for an index buffer. So create yet another class variable where we declared `vertexBuffer` with the line below:

```
private IndexBuffer indexBuffer;
```

Next, create a method called `SetupIndexBuffer()` that looks like the line below:

```
protected void SetupIndexBuffer()
```

We need to create the data for our index buffer. You can use the lines of code below to do this. Put them in the SetupIndexBuffer() method we just created.

```
short[] indices = new short[60];
indices[0] = 0;      indices[1] = 6;       indices[2] = 1;
indices[3] = 0;      indices[4] = 11;      indices[5] = 6;
indices[6] = 1;      indices[7] = 4;       indices[8] = 0;
indices[9] = 1;      indices[10] = 8;      indices[11] = 4;
indices[12] = 1;     indices[13] = 10;     indices[14] = 8;
indices[15] = 2;     indices[16] = 5;      indices[17] = 3;
indices[18] = 2;     indices[19] = 9;      indices[20] = 5;
indices[21] = 2;     indices[22] = 11;     indices[23] = 9;
indices[24] = 3;     indices[25] = 7;      indices[26] = 2;
indices[27] = 3;     indices[28] = 10;     indices[29] = 7;
indices[30] = 4;     indices[31] = 8;      indices[32] = 5;
indices[33] = 4;     indices[34] = 9;      indices[35] = 0;
indices[36] = 5;     indices[37] = 8;      indices[38] = 3;
indices[39] = 5;     indices[40] = 9;      indices[41] = 4;
indices[42] = 6;     indices[43] = 10;     indices[44] = 1;
indices[45] = 6;     indices[46] = 11;     indices[47] = 7;
indices[48] = 7;     indices[49] = 10;     indices[50] = 6;
indices[51] = 7;     indices[52] = 11;     indices[53] = 2;
indices[54] = 8;     indices[55] = 10;     indices[56] = 3;
indices[57] = 9;     indices[58] = 11;     indices[59] = 0;
```

Now we need to get this data into our index buffer, so add the following lines of code to the bottom of the SetupIndexBuffer() method:

```
indexBuffer = new IndexBuffer(device, typeof(short), 60,
            ResourceUsage.WriteOnly, ResourceManagementMode.Automatic);
indexBuffer.SetData(indices);
```

The first line creates a new IndexBuffer object, giving it the graphics device we are drawing with, the type of data that is being passed into it, the number items being passed to it, and a couple of constants that tell the index buffer how to treat the data.

The second line actually copies the data into the index buffer.

Drawing with an Index Buffer

Now that we have created our index buffer and our vertex buffer, let's make our program use these to draw. The first thing we need to do is call the methods we've defined. Add the lines below to the Initialize() method:

```
            SetupVertexBuffer();
            SetupIndexBuffer();
```

Now, go to the `Draw()` method. We are going to make some major changes here. In between the `pass.Begin();` and the `pass.End();` lines are two lines of code. We want to delete these lines of code, because we are going to be doing our drawing a little differently.

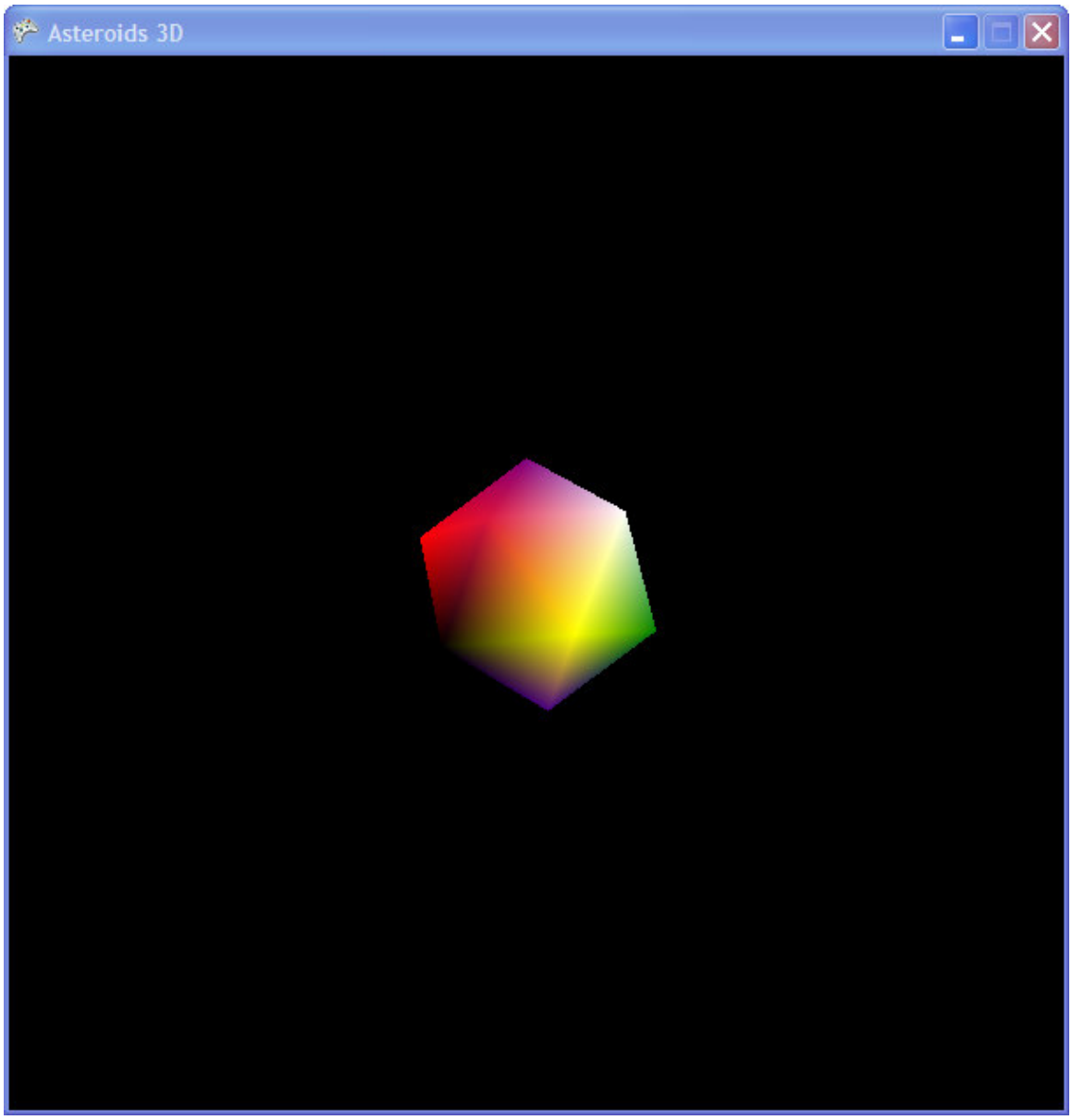
In their place, put the following four lines of code:

```
device.Vertices[0].SetSource(vertexBuffer, 0,
                  VertexPositionColor.SizeInBytes);
device.Indices = indexBuffer;
device.VertexDeclaration = new VertexDeclaration(device,
                  VertexPositionColor.VertexElements);
device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, 12, 0,
                  20);
```

The first two lines of code tell the graphics card what data to use. The first line gives it our vertex buffer, along with an initial offset, and a vertex stride (how far to move ahead between vertices). The second line gives it our index buffer.

The third line and fourth lines are quite a bit like the lines we deleted. Instead of drawing "user" primitives, we are drawing indexed primitives. The parameters for this method are first, what type of objects we are drawing (a list of triangles), the base vertex (where our vertex list starts in the vertex array), the minimum index (where our list of indices starts), the total number of vertices available, the starting index, and finally, the number of triangles to draw. For what we are doing now, the values that are 0's will almost always be 0. You will only need to change the number of vertices (the 12) and the number of triangles (the 20) in order to do different objects.

We should now be able to run our program again to see the output. We should see something like the image below:


Asteroids 3D